IOMMU Event Tracing – What It Is and How It Can Help Your Distro?

Shuah Khan – Sr. Linux Kernel Developer
Open Source Innovation Group
Samsung Research America (Silicon Valley)
shuahkh@osg.samsung.com

© 2015 SAMSUNG Electronics Co.

Abstract

IOMMU event tracing feature enables reporting IOMMU events as they happen during boot-time and run-time. As an example, when a device is detached from host and assigned to a virtual machine, the device gets moved from host domain to vm domain.

Enabling IOMMU event tracing will provide useful information about the devices that are using IOMMU as well as as the changes that occur in device assignments. In this talk, we will discuss the IOMMU event tracing feature and how to enable and use it to trace events during boot-time and run-time. The discussion will be focused on using the IOMMU tracing feature to get insight into what's happening on a system in virtualized environments as devices get assigned from host to virtual machines and vice versa. Linux kernel developers and users can learn about a feature that can aid during development, maintenance, and support of systems with IOMMU.

© 2015 SAMSUNG Electronics Co.

Agenda

- What is an IOMMU?
- What does IOMMU do for us?
- IOMMU references
- IOMMU groups – device isolation
- IOMMU domains - protection
- IOMMU Event Tracing – classes
- IOMMU Event Tracing – group class events
- IOMMU Event Tracing – device class events
- IOMMU Event Tracing – map and unmap events
- IOMMU Event Tracing - error class events
- How to enable IOMMU Event Tracing at boot-time?
- How to enable IOMMU Event Tracing at run-time?
- Where are those traces?
- What do IOMMU group event traces look like?
- What does lspci show?
- IOMMU groups and device topology
- What do IOMMU device event traces look like?
- What do IOMMU map and unmap event traces look like?
- Great we have traces! What now? Using traces to solve problems
- VFIO based device assignment use-case
- Result - VFIO patch series to fix problems!
- Result - Improvements to IOMMU tracing feature

© 2015 SAMSUNG Electronics Co.

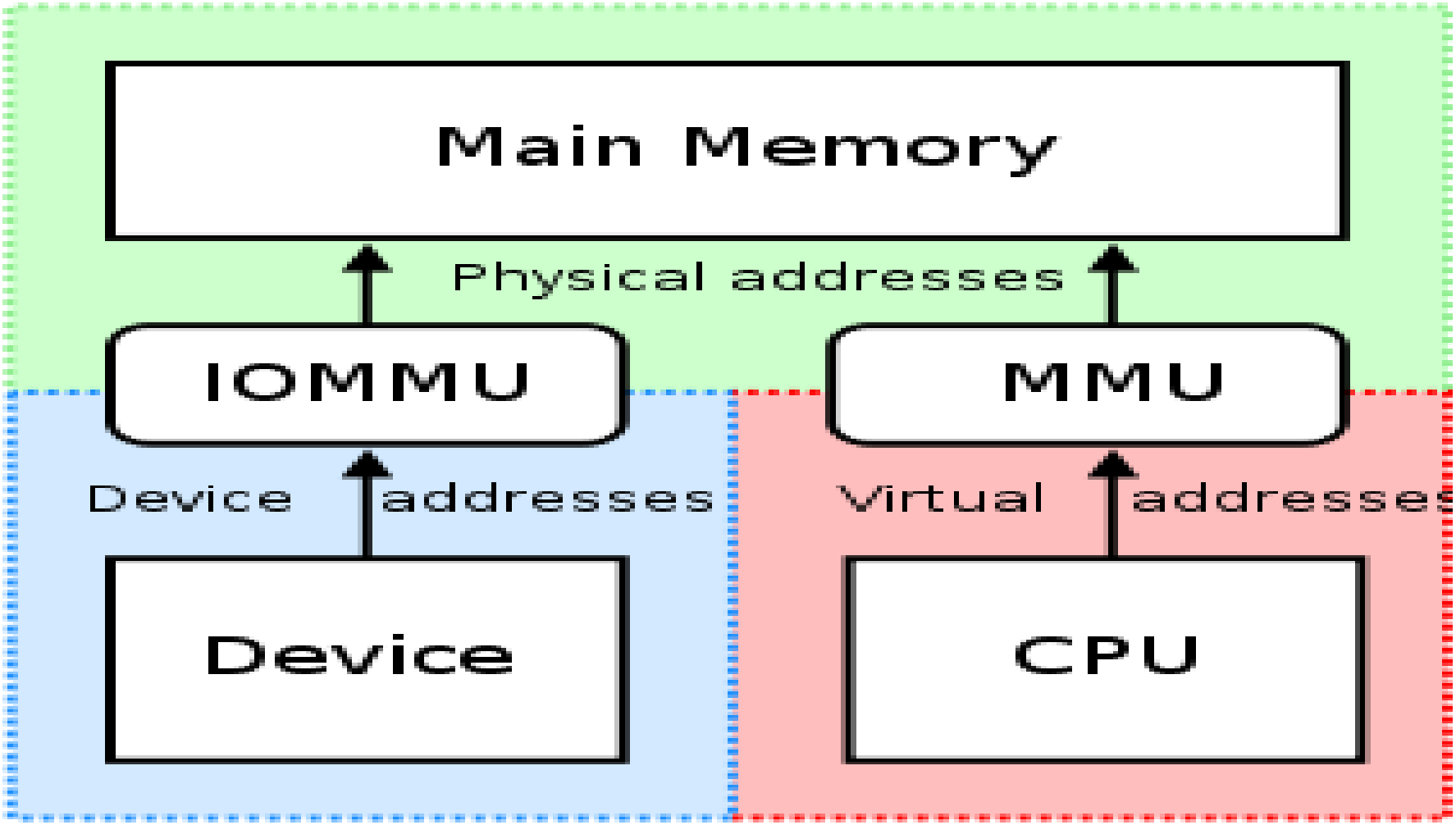
What is an IOMMU?

I/O Memory Management Unit:

- Translation - maps device (I/O) address to physical (machine) address.
- Isolation - device isolation via access permissions (allow/disallow access to memory regions or grant/deny map requests).
- I/O Virtualization - virtual address space (iova)
  - Each I/O device is assigned a DMA virtual address space same as physical address space or virtual address space.

© 2015 SAMSUNG Electronics Co.

IO Memory Management Unit – maps device addresses to physical addresses



© 2015 SAMSUNG Electronics Co.

What does IOMMU do for us?

Advantages:

- One single contiguous virtual memory region can be mapped to multiple non-contiguous physical memory regions. IOMMU can make a non-contiguous memory region appear contiguous to a device (scatter/gather).
- Scatter/gather optimizes streaming DMA performance for the I/O device
- Memory isolation and protection: device can only access memory regions that are mapped for it.
  - Hence faulty and/or malicious devices can't corrupt memory.
- Memory isolation allows safe device assignment to a virtual machine without compromising host and other guest OSes.
- IOMMU enables 32-bit DMA capable non-DAC devices access to > 4GB memory.
- IOMMU - support hardware interrupt re-mapping.
  - extends limited hardware interrupts to software interrupts.
  - interrupt remapping - primary uses are interrupt isolation and translation between interrupt domains, ex. ioapic vs x2apic on x86

Disadvantages:

- Latency in dynamic DMA mapping path, translation over head penalty.
- IOTLB can alleviate translation overhead and most servers support IOMMU and IOTLB hardware.

© 2015 SAMSUNG Electronics Co.

IOMMU groups – device isolation

- Single device isolation is not possible in some cases for variety of reasons.
  - e.g: Devices behind bridge can communicate without reaching IOMMU
  - Multi-function cards don't always support PCI access control services required to describe isolation between functions.
- Devices are grouped for isolation in IOMMU groups.
- Each group contains devices that should be isolated as a group, when single device granularity isn't possible.

© 2015 SAMSUNG Electronics Co.

Device isolation at port granularity – Not!!!

IOMMU

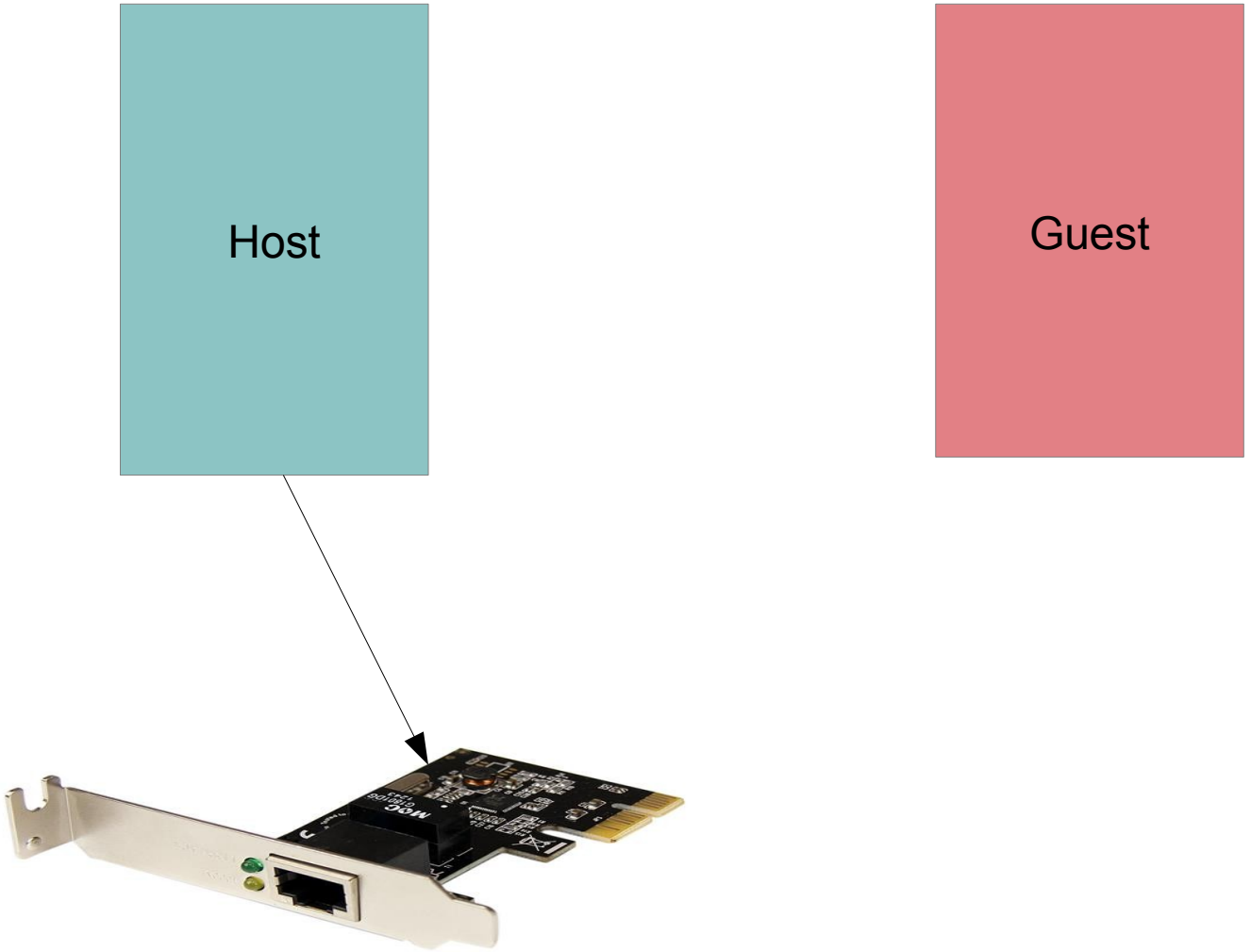
© 2015 SAMSUNG Electronics Co.

IOMMU domains - protection

- Domains provide protection against one guest VM corrupting another VM's memory.
- Devices get moved from one domain to another when a device gets moved from one VM to another or host to a guest.

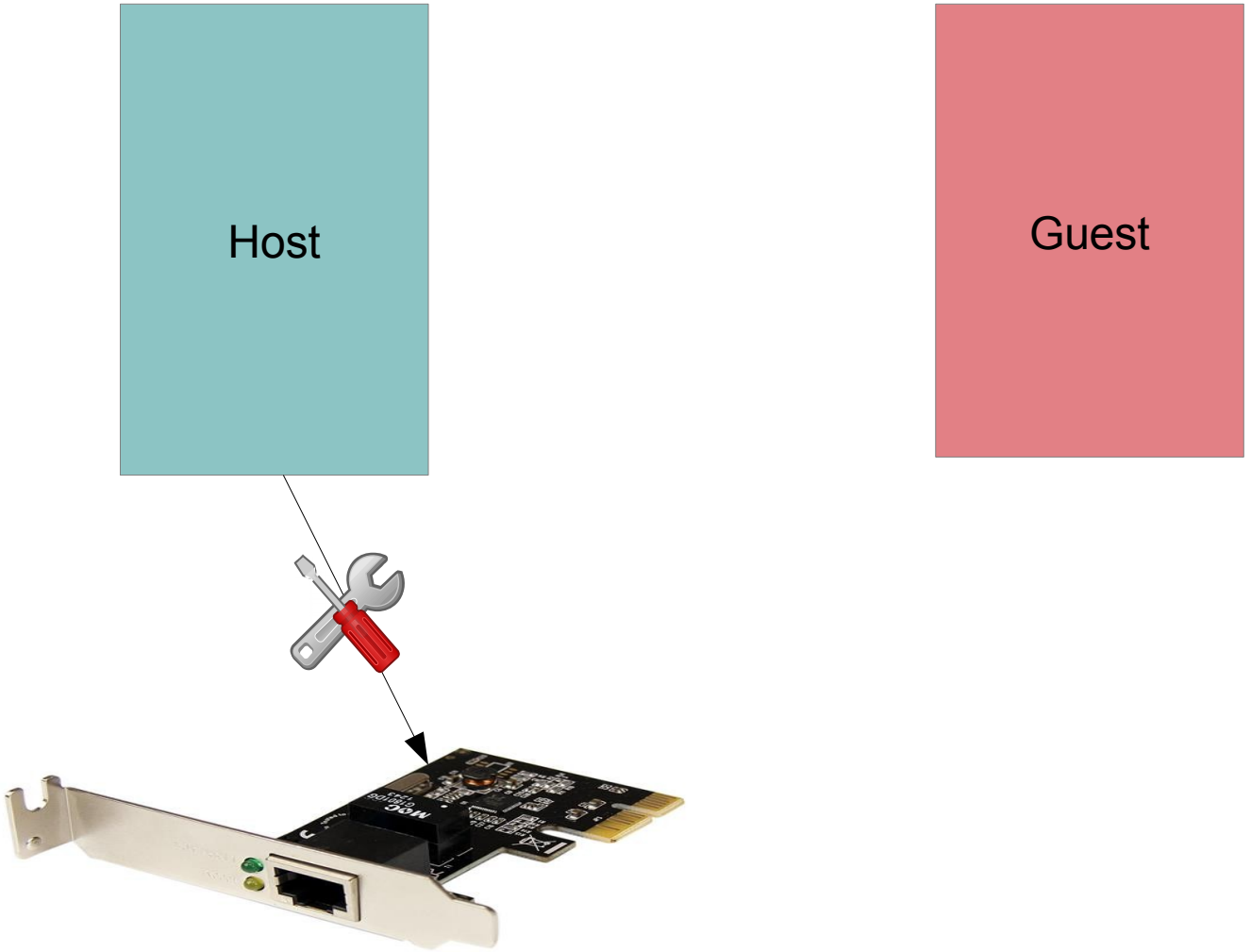
© 2015 SAMSUNG Electronics Co.

Device assigned to host



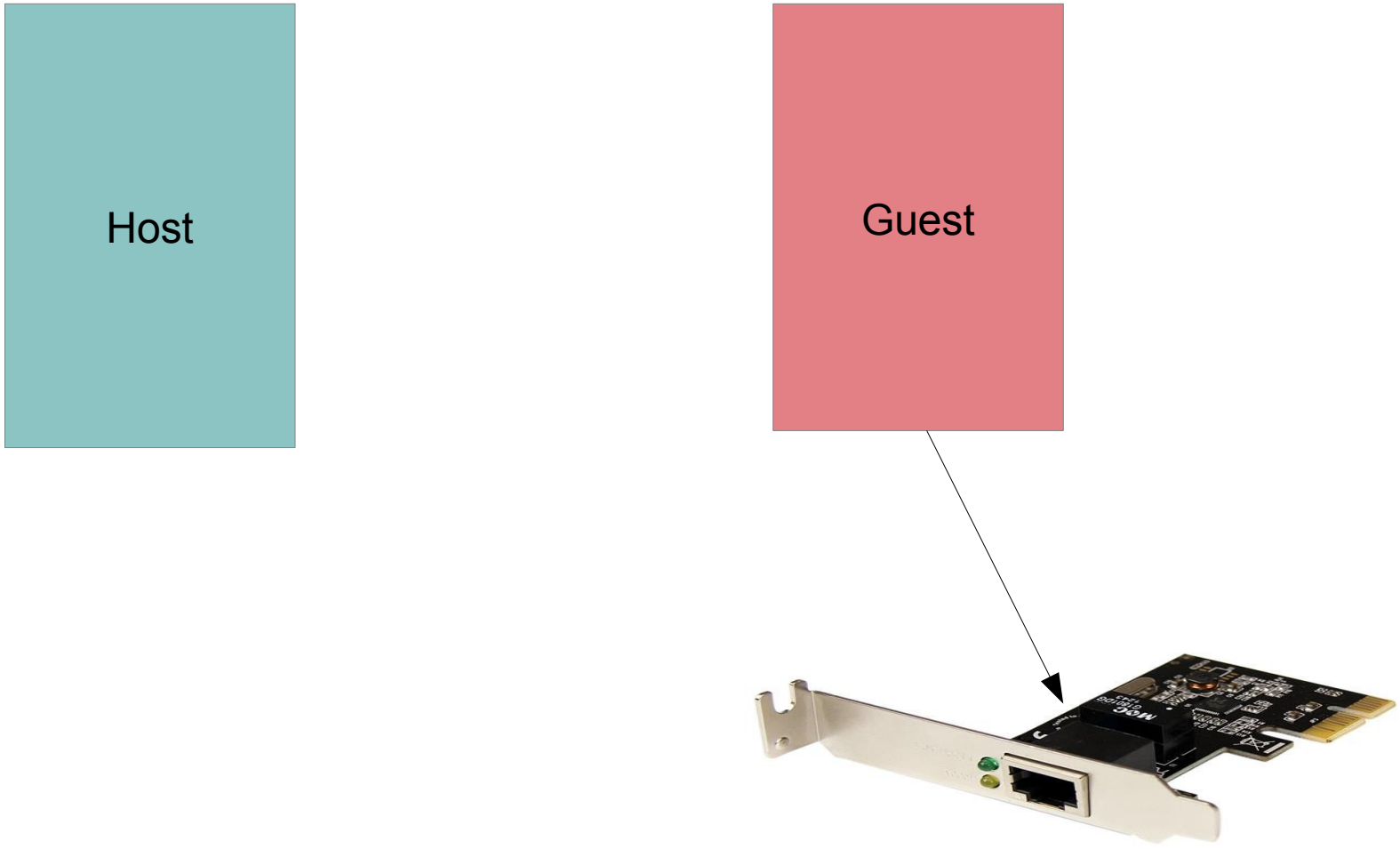
© 2015 SAMSUNG Electronics Co.

Device detached from host



© 2015 SAMSUNG Electronics Co.

Device assigned to guest



© 2015 SAMSUNG Electronics Co.

IOMMU Event Tracing - classes

- IOMMU group class events:
  - Add device to IOMMU group.
  - Remove device from IOMMU group.
- IOMMU device class events:
  - Attach device to a domain.
  - Detach device from a domain.
- IOMMU map event.
- IOMMU unmap event.
- IOMMU Error class:
  - io_page_fault event.

© 2015 SAMSUNG Electronics Co.

IOMMU Event Tracing – group class events

- Add device to a group:
  - Format: IOMMU: groupID=%d device=%s
- Remove device from a group:
  - Format: IOMMU: groupID=%d device=%s

- Events in this group are triggered during boot.
- This information provides insight into IOMMU device topology and device grouping.

© 2015 SAMSUNG Electronics Co.

IOMMU Event Tracing – device class events

- Attach (add) device to a domain:
  - Format: IOMMU: device=%s
- Detach (remove) device from a domain:
  - Format: IOMMU: device=%s

- Events in this group are triggered during run-time whenever devices are attached to and detached from domains. e.g: When a device is detached from host and attached to a guest.
- This information provides insight into device assignment changes during run-time.

© 2015 SAMSUNG Electronics Co.

IOMMU Event Tracing – map and unmap events

- IOMMU Map:
  - Format: IOMMU: iova=0x%016llx paddr=0x%016llx size=%zu
- IOMMU Unmap:
  - Format: IOMMU: iova=0x%016llx size=%zu unmapped_size=%zu

- Events in this group are triggered during run-time whenever device drivers make IOMMU map and unmap requests.
- This information provides insight into map and unmap requests and helps debug performance and other problems.

© 2015 SAMSUNG Electronics Co.

IOMMU Event Tracing – error class events

- IO Page Fault (AMD-Vi)
  - Format: IOMMU:%s %s iova=0x%016llx flags=0x%04x

- Events in this group are triggered during run-time when an IOMMU fault occurs.
- This information provides insight into IOMMU faults and useful in logging the fault and take measures to restart the faulting device. The information in flags field is especially useful in debugging IOMMU kernel

© 2015 SAMSUNG Electronics Co.

How to enable IOMMU tracing at boot-time?

Using Kernel boot option trace_event:

The following enables all IOMMU trace events at boot-time.

trace_event=iommu

© 2015 SAMSUNG Electronics Co.

How to enable IOMMU tracing at run-time?

Enable single event:

```
cd /sys/kernel/debug/trace/events
echo 1 > iommu/event_name_file
```

or

Enable all events:

```
for i in $(find /sys/kernel/debug/tracing/events/iommu/ -name enable);
do echo 1 > $i; done
```


© 2015 SAMSUNG Electronics Co.

Where are those traces?

/sys/kernel/debug/tracing/trace

```
# tracer: nop
#
# entries-in-buffer/entries-written: 18/18   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
```


© 2015 SAMSUNG Electronics Co.

What do IOMMU group event traces look like?

```
# tracer: nop
#
# entries-in-buffer/entries-written: 18/18   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||   TIMESTAMP  FUNCTION
#              | |       |   ||||      |         |
      swapper/0-1     [000] ....     1.899609: add_device_to_group: IOMMU: groupID=0 device=0000:00:00.0
      swapper/0-1     [000] ....     1.899619: add_device_to_group: IOMMU: groupID=1 device=0000:00:01.0
      swapper/0-1     [000] ....     1.899624: add_device_to_group: IOMMU: groupID=2 device=0000:00:02.0
      swapper/0-1     [000] ....     1.899629: add_device_to_group: IOMMU: groupID=3 device=0000:00:03.0
      swapper/0-1     [000] ....     1.899634: add_device_to_group: IOMMU: groupID=4 device=0000:00:14.0
      swapper/0-1     [000] ....     1.899642: add_device_to_group: IOMMU: groupID=5 device=0000:00:16.0
      swapper/0-1     [000] ....     1.899647: add_device_to_group: IOMMU: groupID=6 device=0000:00:1a.0
      swapper/0-1     [000] ....     1.899651: add_device_to_group: IOMMU: groupID=7 device=0000:00:1b.0
      swapper/0-1     [000] ....     1.899656: add_device_to_group: IOMMU: groupID=8 device=0000:00:1c.0
      swapper/0-1     [000] ....     1.899661: add_device_to_group: IOMMU: groupID=9 device=0000:00:1c.2
      swapper/0-1     [000] ....     1.899668: add_device_to_group: IOMMU: groupID=10 device=0000:00:1c.3
      swapper/0-1     [000] ....     1.899674: add_device_to_group: IOMMU: groupID=11 device=0000:00:1d.0
      swapper/0-1     [000] ....     1.899682: add_device_to_group: IOMMU: groupID=12 device=0000:00:1f.0
      swapper/0-1     [000] ....     1.899687: add_device_to_group: IOMMU: groupID=12 device=0000:00:1f.2
      swapper/0-1     [000] ....     1.899692: add_device_to_group: IOMMU: groupID=12 device=0000:00:1f.3
      swapper/0-1     [000] ....     1.899696: add_device_to_group: IOMMU: groupID=13 device=0000:02:00.0
      swapper/0-1     [000] ....     1.899701: add_device_to_group: IOMMU: groupID=14 device=0000:03:00.0
      swapper/0-1     [000] ....     1.899704: add_device_to_group: IOMMU: groupID=10 device=0000:04:00.0
```

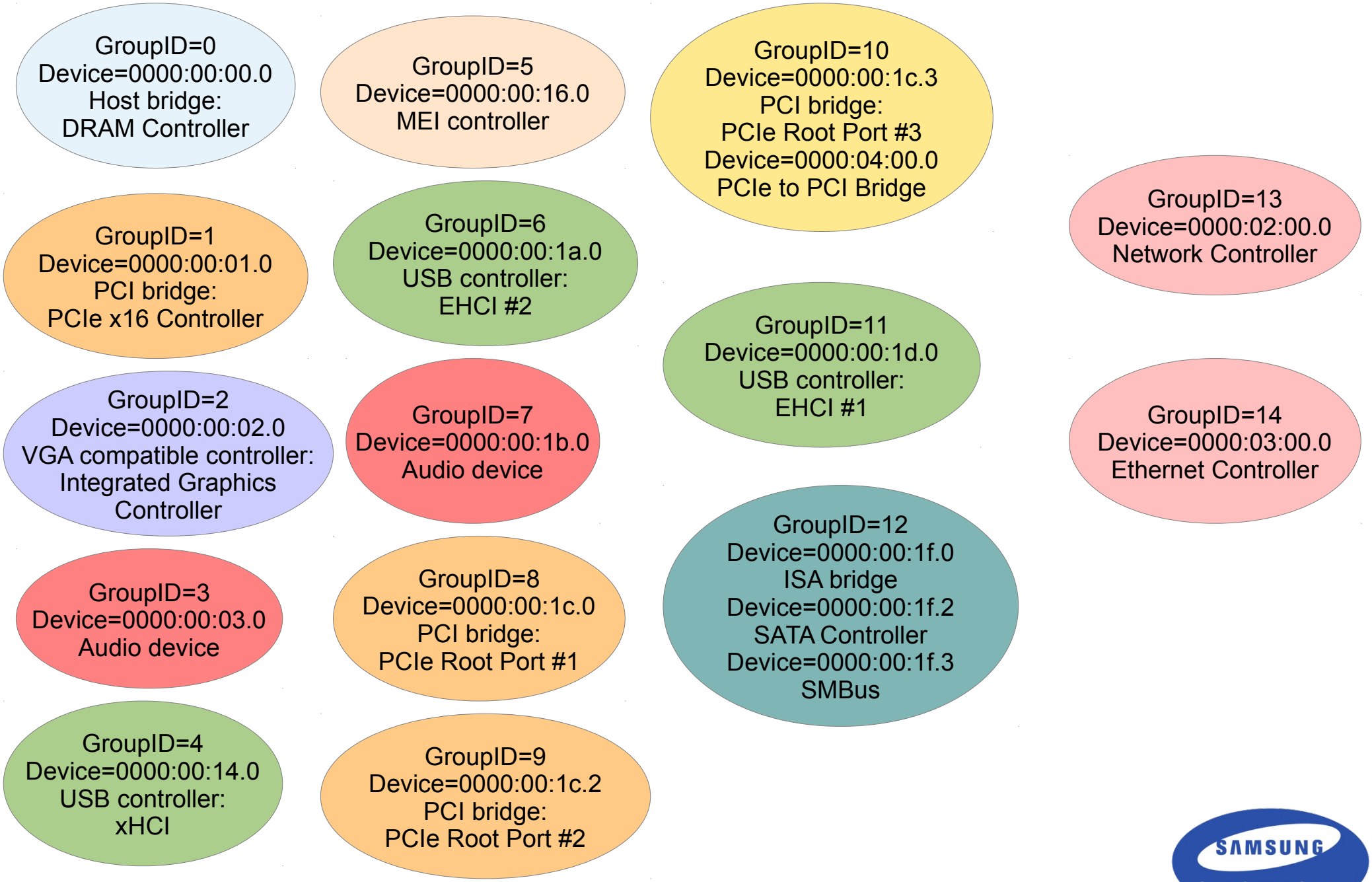

© 2015 SAMSUNG Electronics Co.

What does lspci show?

00:00.0 Host bridge: Intel Corporation 4th Gen Core Processor DRAM Controller (rev 06)
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor PCI Express x16 Controller (rev 06)
00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor Integrated Graphics Controller (rev 06)
00:03.0 Audio device: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor HD Audio Controller (rev 06)
00:14.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB xHCI (rev 05)
00:16.0 Communication controller: Intel Corporation 8 Series/C220 Series Chipset Family MEI Controller #1 (rev 04)
00:1a.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI #2 (rev 05)
00:1b.0 Audio device: Intel Corporation 8 Series/C220 Series Chipset High Definition Audio Controller (rev 05)
00:1c.0 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #1 (rev d5)
00:1c.2 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #3 (rev d5)
00:1c.3 PCI bridge: Intel Corporation 82801 PCI Bridge (rev d5)
00:1d.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI #1 (rev 05)
00:1f.0 ISA bridge: Intel Corporation H87 Express LPC Controller (rev 05)
00:1f.2 SATA controller: Intel Corporation 8 Series/C220 Series Chipset Family 6-port SATA Controller 1 [AHCI mode] (rev 05)
00:1f.3 SMBus: Intel Corporation 8 Series/C220 Series Chipset Family SMBus Controller (rev 05)
02:00.0 Network controller: Intel Corporation Wireless 7260 (rev 73)
03:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
04:00.0 PCI bridge: ASMedia Technology Inc. ASM1083/1085 PCIe to PCI Bridge (rev 04)

© 2015 SAMSUNG Electronics Co.

IOMMU groups and device topology



© 2015 SAMSUNG Electronics Co.

What do IOMMU device event traces look like?

```
# tracer: nop
#
# entries-in-buffer/entries-written: 5689868/5689868   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
    qemu-kvm-28546 [003] ....  1804.692631: attach_device_to_domain: IOMMU: device=0000:00:1c.0
    qemu-kvm-28546 [003] ....  1804.692635: attach_device_to_domain: IOMMU: device=0000:00:1c.4
    qemu-kvm-28546 [003] ....  1804.692643: attach_device_to_domain: IOMMU: device=0000:05:00.0
    qemu-kvm-28546 [003] ....  1804.692666: detach_device_from_domain: IOMMU: device=0000:00:1c.0
    qemu-kvm-28546 [003] ....  1804.692671: detach_device_from_domain: IOMMU: device=0000:00:1c.4
    qemu-kvm-28546 [003] ....  1804.692676: detach_device_from_domain: IOMMU: device=0000:05:00.0
```


© 2015 SAMSUNG Electronics Co.

What do IOMMU map/unmap event traces look like?

```
# tracer: nop
#
# entries-in-buffer/entries-written: 54/54   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
qemu-kvm-28546 [002] ....  1804.480679: map: IOMMU: iova=0x00000000000a0000
    paddr=0x00000000446a0000 size=4096
qemu-kvm-28547 [006] ....  1809.032767: unmap: IOMMU: iova=0x00000000000c1000
    size=4096 unmapped_size=4096
```


© 2015 SAMSUNG Electronics Co.

Great we have traces! What now?
Using traces to solve problems...

© 2015 SAMSUNG Electronics Co.

Using traces -----

Get insight into:

- IOMMU device topology – which devices belong to which group
- Run-time device assignment changes as devices move from host to guests and back to host.

Debug:

- IOMMU problems.
- Device assignment problems.
- Detect and solve performance problems.
- BIOS and firmware problems related to IOMMU hardware and firmware implementation.

© 2015 SAMSUNG Electronics Co.

VFIO based device assignment use-case

Alex Williamson enabled run-time IOMMU traces for vfio-based device assignment and found the following VFIO problems:

- Large number of unmap calls on VT-d system without IOMMU superpage support:
  - VFIO unmap path is not optimized on a VT-d system without IOMMU superpage support: each single page is unmapped individually, since the current unmap path optimization relies on IOMMU superpage support.
- Unnecessary single page mappings for invalid and reserved memory regions, like mappings of MMIO BARs.
- Very long task runs with needs-resched set.

© 2015 SAMSUNG Electronics Co.

Result - VFIO patch series to fix problems!

Alex was able to:

- Reduce the number of unmap calls to 2% of the original on Intel VT-d without IOMMU superpage support.
  - Before: maps 472,574, unmaps 5,217,244 – unmaps are 10+ times the number of maps.
  - After: maps 9509, unmaps 9509
- Sporadic needs-resched runs.

Reference: http://lists.linuxfoundation.org/pipermail/iommu/2015-January/011718.html

© 2015 SAMSUNG Electronics Co.

Result - Improvements to IOMMU tracing feature

Alex found a few bugs and suggested improvements:

- trace_iommu_map() should report original iova and size.
- trace_iommu_unmap() should report original iova, size, and unmapped size.
- Size field is handled as int and could overflow.
- The above problems are fixed in 3.20
  - iommu: fix trace_map() to report original iova and original size
  - iommu: fix trace_unmap() to report original iova
  - iommu: change trace unmap api to report unmapped size

© 2015 SAMSUNG Electronics Co.

Acknowledgements

Special thanks to Alex Williamson:

- for generating traces for VFIO based device assignments.
- for his feedback on improving the IOMMU Event Tracing API.

© 2015 SAMSUNG Electronics Co.

IOMMU References

- Utilizing IOMMUs for Virtualization in Linux and Xen, Multiple Authors
- https://www.kernel.org/doc/Documentation/vfio.txt
- VFIO PCI Device assignment breaks free of KVM – Alex Williamson, RedHat

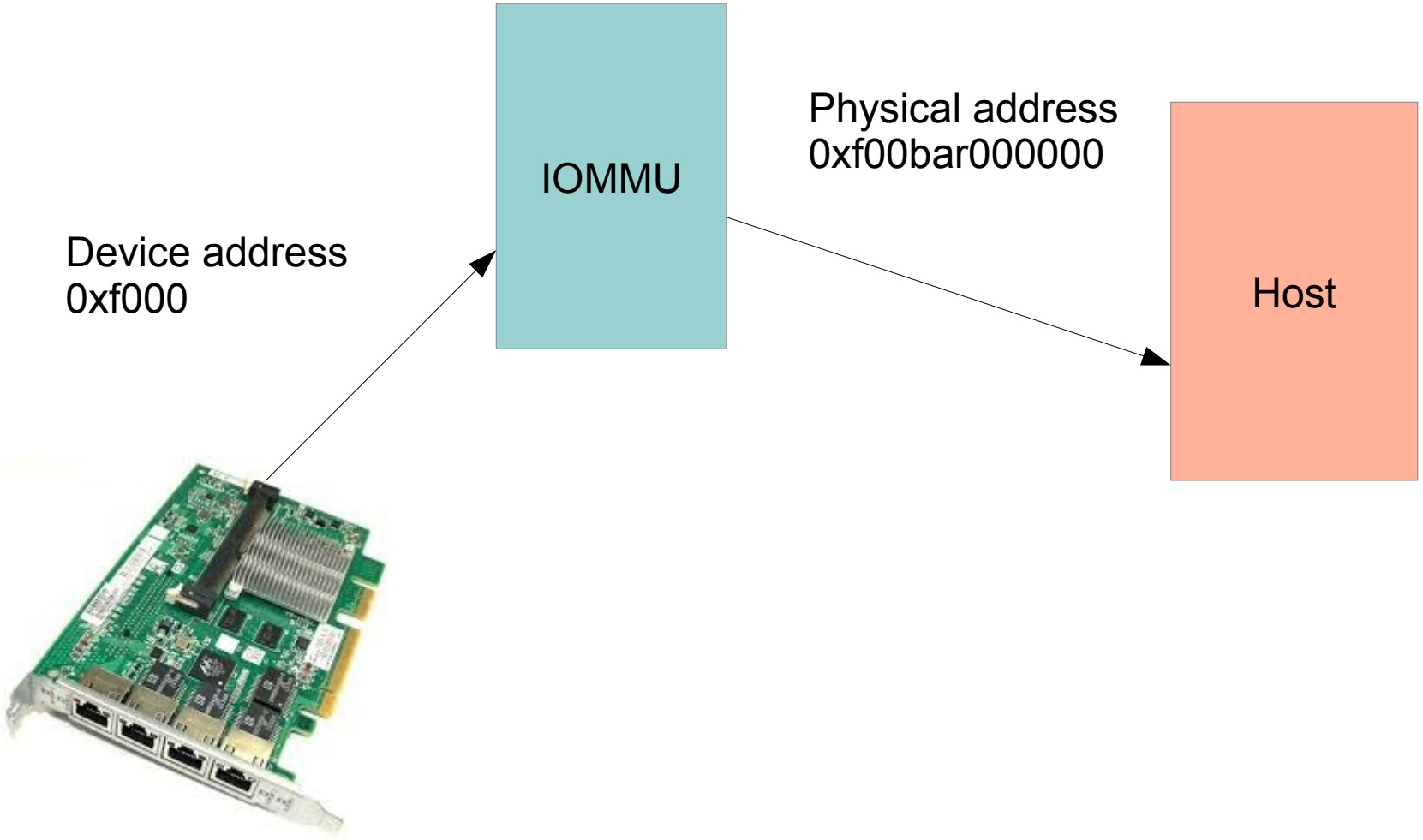
© 2015 SAMSUNG Electronics Co.

SAMSUNG
Open Source Group

Thank you.

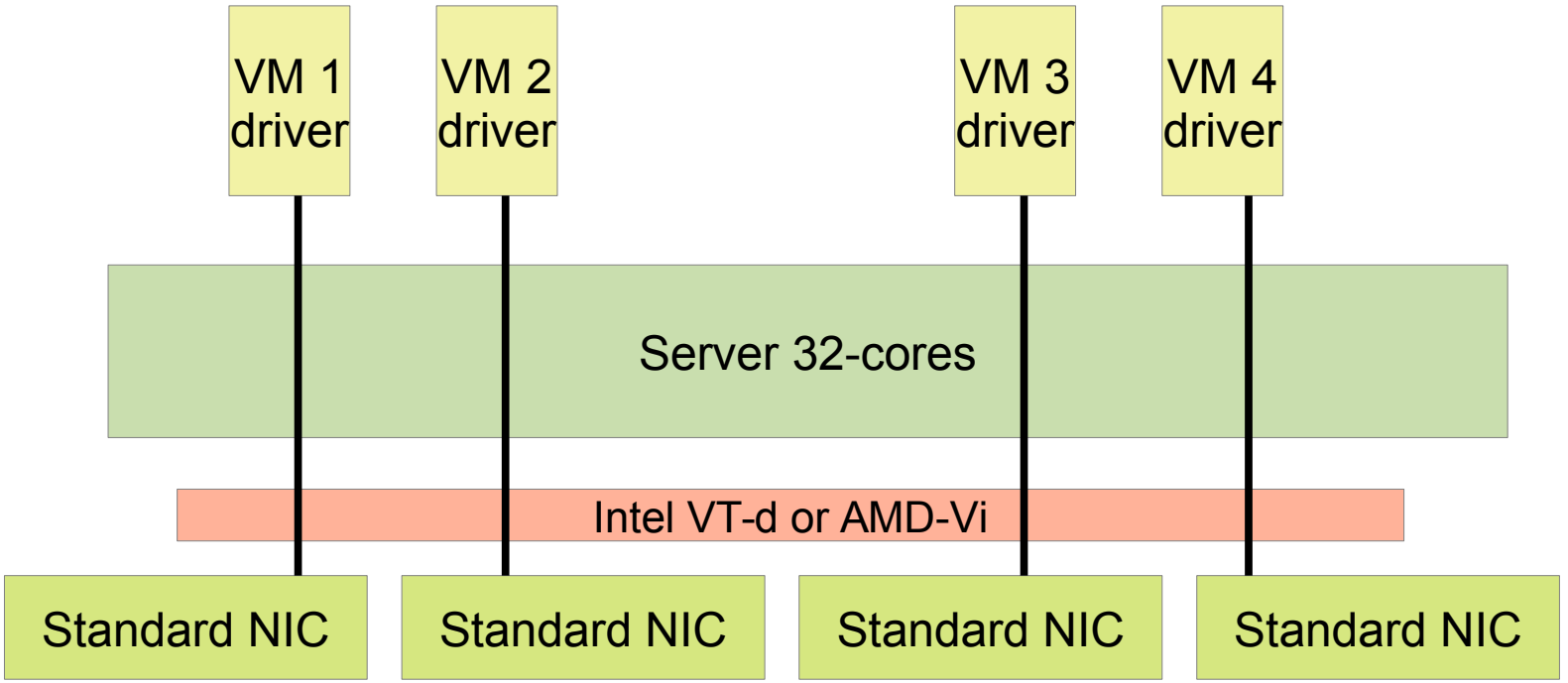
© 2015 SAMSUNG Electronics Co.

IOMMU lookups



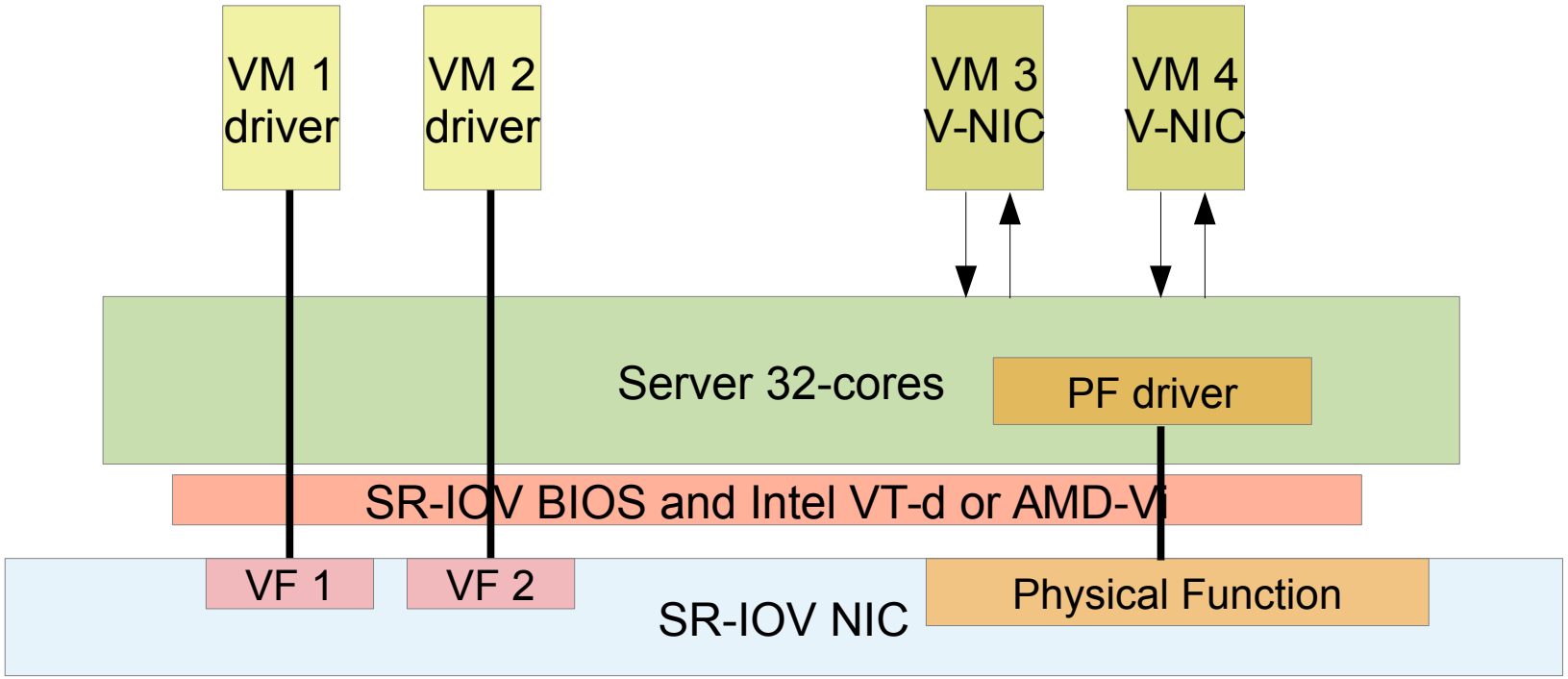
© 2014 SAMSUNG Electronics Co.

Physical Device Assignment



© 2014 SAMSUNG Electronics Co.

Virtual Device Assignment



© 2014 SAMSUNG Electronics Co.